Knowledge-Level Planning for Task-Based Social Interaction

Knowledge-Level Planning for Task-Based Social Interaction

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Mary Ellen Foster
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
M.E.Foster@hw.ac.uk

Abstract

A robot coexisting with humans must not only be able to perform physical tasks, but must also be able to interact with humans in a socially appropriate manner. In many social settings, this involves the use of social signals like gaze, facial expression, and language. In this paper, we discuss the problem of planning social and task-based actions for a robot that must interact with multiple human agents in a dynamic domain. We show how social states are inferred from low-level sensors, using vision and speech as input modalities, and use a general purpose knowledge-level planner to model task, dialogue, and social actions, as an alternative to current mainstream methods of interaction management. The resulting system has been evaluated in a real-world study with human subjects, in a simple bartending scenario.

Introduction and Motivation

As robots become integrated into daily life, they must increasingly deal with situations in which *socially appropriate interaction* is vital. In such settings, it is not enough for a robot simply to achieve task-based goals; instead, it must also be able to satisfy the social goals and obligations that arise through interactions with people in real-world settings. However, the problem of building a robot to meet the goals of social interaction presents several challenges, especially for the reasoning and action selection components of such a system. Not only does the robot require the ability to recognise and understand appropriate multimodal social signals (e.g., gaze, facial expression, and language), but it must also generate realistic responses using similar modalities.

To address this challenge, and help focus our work, we are investigating the sub-problem of *task-based social interaction* using a bartending scenario as our target domain. In particular, we are developing a robot bartender (Figure 1) that is capable of dealing with multiple customers in a drink-ordering scenario. Interactions in this scenario incorporate both task-based aspects (e.g., ordering and paying for drinks) and social aspects (e.g., managing multiple interactions). Moreover, the primary interaction modality in this setting is speech; humans communicate with the robot via speech and the robot must respond in a similar manner.

Our approach to high-level reasoning and action selection uses AI planning, specifically, *knowledge-level planning* (Petrick and Bacchus 2002; 2004) techniques. The ability to reason and plan is essential for a cognitive agent acting in a dynamic and incompletely known world such as the bartending scenario. General-purpose automated planners are good at building goal-directed plans of action under many challenging conditions, especially in task-based contexts. Moreover, recent work (Steedman and Petrick 2007; Brenner and Kruijff-Korbayová 2008; Benotti 2008; Koller and Petrick 2011) has investigated the use of automated planning for natural language generation and dialogue—research fields that have a long tradition of using planning, but where such techniques are no longer the focus of mainstream study. The use of planning for natural language processing is particularly important since plan generation in the bartending domain will require a mix of traditional task-based actions (e.g., handing the customer a drink) and speech acts (e.g., asking a customer for a drink order).

Figure 1: The robot bartender

While planning offers a tool for reasoning and action selection, it is only one component in a larger robot system that must operate in a real-world environment. This introduces some difficulties that must be overcome when the planner interacts with other parts of the system. For instance, the problem of integrating low-level sensor data with symbolic planners introduces representational difficulties that must be addressed: high-level planners typically use representations based on discrete models of objects, properties, and actions, described in logical languages, while many low-level sensors tend to generate continuous streams of low-level, noisy data. Moreover, some aspects of the traditional planning problem, like the initial state, cannot be defined

A customer approaches the bar and looks at the bartender	
ROBOT:	[Looks at Customer 1] How can I help you?
CUSTOMER 1:	A pint of cider, please.
Another customer approaches the bar and looks at the bartender	
ROBOT:	[Looks at Customer 2] One moment, please.
ROBOT:	[Serves Customer 1]
ROBOT:	[Looks at Customer 2]
	Thanks for waiting. How can I help you?
CUSTOMER 2:	I'd like a pint of beer.
ROBOT:	[Serves Customer 2]

Figure 2: An example interaction in the bartending scenario.

offline (e.g., the number of customers in the bar). Instead, they must be provided to the planner based on observations of the scene sensed by low-level input modalities such as vision and speech. Furthermore, in an inherently dynamic domain like a busy bar, these sensors may not be able to fully observe the world, or may provide noisy information. Thus, the planner cannot be viewed as simply a black box but must be appropriately situated in the wider cognitive system.

We focus on three main areas of work in this paper:

- First, we describe the structure of the robot system and highlight the role of the planner within the architecture. In particular, we focus on how states are derived from sensor observations and how generated plans are executed.
- Second, we show how plans are built by viewing task-based social interaction as an instance of planning with incomplete information and sensing, using the PKS planner (Petrick and Bacchus 2002; 2004) as an alternative to more mainstream methods of interaction management.
- Finally, we present a planning domain for a simple bartending scenario, and show how it generates interesting interaction behaviours. This domain has been evaluated in a real-world study, and provides the basis for future work.

This work forms part of a larger project called JAMES, Joint Action for Multimodal Embodied Social Systems, exploring social interaction with robot systems.[1]

Overview of the Scenario and System

The target application for this work is a bartending scenario supporting interactions similar to the one in Figure 2. In this scenario, two agents enter the bar area and attempt to attract the robot's attention to order a drink. Even this simple interaction presents certain challenges to the robot system tasked with the role of bartender: the vision system must track the locations and body postures of the agents; the speech-recognition system must detect and deal with speech in an open setting; the reasoning components must determine that both customers require attention and should ensure that they are served in the order that they arrived; while the output components must select and execute concrete actions for each output channel that correctly realises high-level plans.

To address these challenges, the system architecture builds on a standard three-layer structure: the low level deals

[1] See `http://james-project.eu/` for more information.

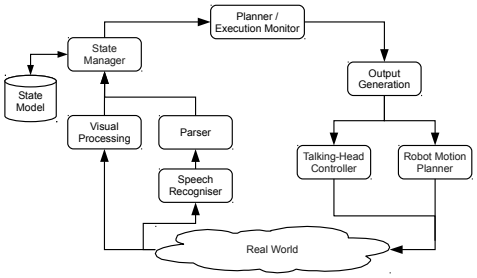

Figure 3: System architecture

with modality-specific information which is often continuous (e.g., spatial coordinates and speech-recognition hypotheses); the mid-level works with more general, cross-modality representations of states and events (e.g., agent intentions); while the high level reasons about abstract structures in a logical form (e.g., knowledge and action).

Robot hardware and vision system: The robot platform provides the sensors and effectors that interact with the real world. The robot hardware itself consists of two 6-degrees-of-freedom industrial manipulator arms with grippers, mounted to resemble human arms. Sitting on top of the main robot torso is a Philips iCat animatronic talking head capable of producing facial expressions, rigid head motion, and lip-synchronised synthesised speech.

One of the primary input modalities used by the robot is vision. The vision system tracks the location, facial expressions, gaze behaviour, and body language of all people in the scene in real time. This done by using input from visual sensors to detect and track the faces and hands of agents in the scene, and to extract their 3D position (Baltzakis, Pateraki, and Trahanias 2012). Each agent's focus of attention is also derived using torso orientation. The partially abstracted information resulting from this process is then made available to modules like the state manager for further processing.

Natural language understanding: A second important input modality in the system is speech, since linguistic interaction is central to social communication. In our case, the linguistic processing system combines a speech recogniser with a natural-language parser to create symbolic representations of the speech produced by all users. For speech recognition, the Microsoft Kinect and the associated Microsoft Speech API is used. For a user's utterance u, the system provides a list of intermediate hypotheses and associated confidence scores $\Pi_u = \{\langle h_1, c_1\rangle, \langle h_2, c_2\rangle, \ldots \langle h_n, c_n\rangle\}$, where $\sum_{i=1}^{n} c_i = 1$, a final best hypothesis h_u^* along with a confidence score c_u^*, and an estimate of the source angle, θ_u^*. We have also created a speech recognition grammar that covers all expected user utterances in the bartending scenario. The resulting grammar constrains the recognition task, producing more reliable results. (Currently, only the top hypothesis with its confidence score and angle is used.)

Once the user's speech has been recognised, it must be further processed to extract the underlying meaning. To do this, we parse the recognised speech hypothesis using a grammar defined in OpenCCG (White 2006). OpenCCG is an open-source implementation of Combinatory Categorial Grammar (Steedman 2000), a unification-based categorial

framework which is both linguistically and computationally attractive. The grammar contains both syntactic and semantic information, and is used both for parsing the linguistic input, and for surface realisation of the selected output (see below). Once the top speech hypothesis has been parsed, the logical form, confidence measure, and angle information is passed as an XML structure to the state manager.

State management: The primary role of the state manager is to turn the continuous stream of messages produced by the low-level input components into a discrete representation that combines social and task-based properties. Formally, the low-level input components correspond to a set Σ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor σ_i returns an observation $obs(\sigma_{i,t})$ about the world at a time point t. We denote the set of all sensor readings at t by $\Sigma_t = \{obs(\sigma_{1,t}), obs(\sigma_{2,t}), \ldots, obs(\sigma_{n,t})\}$. Some sensors (e.g., the speech recogniser) may also attach a confidence measure to an observation, indicating its estimated reliability, and capturing the fact that real-world sensors are often noisy.

The state representation is based on a set Φ of fluents, $\Phi = \{f_1, f_2, \ldots, f_m\}$: first-order predicates and functions that denote particular qualities of the world, the robot, and other entities in the domain. We denote the value of a fluent f_i at a time point t by $f_{i,t}$, and the set of all fluent values at t by Φ_t. Fluents are updated in a Markovian fashion, where the value of a fluent at time point $t + 1$ is a function of the current observations and fluent values at time point t, i.e., $f_{i,t+1} = \Gamma_i(\Sigma_t, \Phi_t)$. Typically, the mapping between sensors and fluents is not one-to-one: a sensor may map to zero, one, or many fluents, as appropriate. A state S_t is then a snapshot of the values of all fluents at a time point t, i.e., $S_t = \Phi_t$.

Intuitively, states represent a point of intersection between low-level sensor data and the high-level structures used by components like the planner. Since states are induced from the mapping of sensor observations to fluent values, the challenge of building an effective state manager rests on defining appropriate mapping functions Γ_i. In the context of social robotics, this is the problem of *social signal processing* (Vinciarelli, Pantic, and Bourlard 2009), which has received an increasing amount of attention in recent years. This process is not always straightforward: often, it is not the sensor data at a single time point that determines the value of a fluent, but rather the patterns found in a sequence of signals. Thus, the value of a fluent might combine information from multiple signals and require temporal cross-modal fusion.

In the bartender robot system, we treat each low-level input component as a set of sensors. The linguistic interpreter corresponds to three sensors: two that observe the parsed content of a user's utterance u and its associated confidence score (the pair $\langle h^*_u, c^*_u \rangle$), and another that returns the estimated angle of the sound source (θ^*_u). The vision system also senses a large number of properties about the agents and objects in the world, including the location, face and torso orientation, and body posture, each of which corresponds to a set of individual sensors, again with confidence scores.

Certain low-level output components are also treated as sensors. For example, the robot arms provide information about the start and end of manipulation actions, together with their success or failure, while the speech synthesiser reports the start and end of all system utterances. Modelling output components as sensors allows information from these sources to be included in the derived state, ensuring the current state of interaction is accurately reflected (e.g., the state of turn taking or whether physical actions succeeded).

The actual fluents modelled in the state are defined by the particular requirements of the scenario: we represent all agents in the scene, their locations, torso orientations, attentional states, and drink requests if they have made one. In addition, we also store the coordinates of all sensed entities and other properties from the vision system to enable the low-level output components to access them as necessary.

In the current system, the state manager is rule-based. One set of rules infers user social states (e.g., seeking attention) based on the low-level sensor data, using guidelines derived from a study of human-human interactions in the bartender domain (Huth 2011). The state manager also incorporates rules that map from the logical forms produced by the parser into communicative acts (e.g., drink orders), and that use the source localisation from the speech recogniser together with the vision properties to determine which customer is likely to be speaking. A final set of rules determines when new state reports are published, which helps control turn-taking.

Planning and execution monitoring: The high-level planner is responsible for taking state reports from the state manager and selecting actions to be executed on the robot. A related component, the execution monitor, tracks the execution of planned actions and, in the case of failures or plan divergences, ensures alternative actions are replanned.

Plans are generated using PKS (Planning with Knowledge and Sensing) (Petrick and Bacchus 2002; 2004), a conditional planner that works with incomplete information and sensing actions. PKS operates at the *knowledge level* and reasons about how the planner's knowledge state, rather than the world state, changes due to action. To do this, PKS works with a subset of a first-order logical language and limited inference. While this representation supports features such as functions and run-time variables, these restrictions also mean that certain types of knowledge (e.g., general disjunctive information) cannot be directly modelled in PKS.

PKS is based on a generalisation of STRIPS (Fikes and Nilsson 1971). In STRIPS, the state of the world is modelled by a single database. Actions update this database and, by doing so, update the planner's world model. In PKS, the planner's knowledge state, rather than the world state, is represented by a set of five databases, each of which models a particular type of knowledge, and can be understood in terms of a modal logic of knowledge. Actions can modify any of the databases, which updates the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) it can represent:

$\mathbf{K_f}$: This database is like a STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. K_f is used to model action effects that change the world. K_f can include any ground literal or function (in)equality mapping ℓ, where $\ell \in K_f$ means "the planner knows ℓ."

$\mathbf{K_w}$: This database models the plan-time effects of "binary" sensing actions. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at run time this disjunction will be resolved. PKS uses such information to include conditional branches in a plan, where each branch assumes one of the possible outcomes is true.

$\mathbf{K_v}$: This database stores information about function values that will become known at execution time. In particular, K_v can model the plan-time effects of sensing actions that return constants. K_v can contain any unnested function term f, where $f \in K_v$ means that at plan time the planner "knows the value of f." At execution time the planner will have definite information about f's value. As a result, PKS can use K_v terms as run-time variables in its plans, and can build conditional plan branches when the set of possible mappings for a function is restricted.

$\mathbf{K_x}$: This database models the planner's "exclusive-or" knowledge. Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal.Such formulae represent a common type of disjunctive knowledge common in planning domains, namely that "exactly one of the ℓ_i is true."

(A fifth database called *LCW* is not used in this work.)

PKS's databases can be inspected through a set of *primitive queries* that ask simple questions about the planner's knowledge state: whether facts are (not) known to be true (a query of the form $[\neg]K(\phi)$), whether function values are (not) known (a query $[\neg]K_v(t)$), or if the planner "knows whether" certain properties are true or not (a query $[\neg]K_w(\phi)$). An inference algorithm evaluates primitive queries by checking the contents of the various databases.

An action in PKS is modelled by a set of *preconditions* that query PKS's knowledge state, and a set of *effects* that update the state. Preconditions are simply a list of primitive queries. Effects are described by a collection of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. E.g., $add(K_f, \phi)$ adds ϕ to the K_f database, while $del(K_w, \phi)$ removes ϕ from the K_w database.

PKS constructs plans by reasoning about actions in a simple forward-chaining manner: if the preconditions of an action are satisfied by the planner's knowledge state, then the action's effects are applied to the state to produce a new knowledge state. Planning then continues from the resulting state. PKS can also build plans with branches, by considering the possible outcomes of its K_w and K_v knowledge. Planning continues along each branch until it satisfies the *goal* conditions, also specified as a list of primitive queries.

In addition to the main planner, PKS is aided by an execution monitor which controls replanning. The monitor takes as input a PKS plan, whose execution it tracks, and a state provided by the state manager, denoting the sensed state. The task of the monitor is to assess how close an expected, planned state is to a sensed state in order to determine whether a plan should continue to be executed. To do this, it tries to ensure that a state still permits the next action (or set of actions) in the plan to be executed, by testing an action's preconditions against the current set of (sensed) state properties. In the case of a mismatch, the planner is directed to build a new plan, using the sensed state as its initial state.

Output generation: Output in the system is based on dividing actions selected by the planner into speech, head motions, and arm manipulation behaviours that can be executed by the robot. To do so, we use a structure containing specifications for each of the output modalities (Isard and Matheson 2012). This structure is generated using a rule-based approach, which splits the planned action into its component subparts. The resulting structure is then passed to the particular output module for execution.

On the linguistic side, we use OpenCCG to generate the robot language output, with the same OpenCCG grammar used for input, since it also contains the language necessary for speech output. The language output description is specified in terms of communicative acts based on Rhetorical Structure Theory (RST) (Mann and Thompson 1988). A generation module then translates the RST structure into OpenCCG logical forms, which are sent to the OpenCCG realiser which outputs text strings that can be turned into speech by the robot's animatronic head.

In addition to speech, the robot system also expresses itself through facial expressions, gaze, and arm manipulation actions. These behaviours are coordinated across the various multimodal output channels to ensure that they are synchronised both temporally and spatially. The animatronic head can currently express a number of pre-assigned expressions, and the robot arm can perform tasks like grasping objects to hand over a drink to a customer. For instance, the high-level action `serve(a1,beer)` (i.e., "serve a beer to agent `a1`") is transformed into a specification that results in the robot smiling (an animatronic head facial expression) while handing over a beer (a robot arm manipulation action) and saying to the customer "here is your drink" (speech output).

Planning Interactions in a Bartending Domain

All actions in the bartending scenario (i.e., task and social) are modelled as part of the same underlying planning domain, rather than using specialised tools as is common practice in modern interactive dialogue systems. As a result, our planning domain must capture the necessary properties and actions of the world, agents, and objects.

Domain properties are based on fluents defined in the state manager. In particular, predicates in the domain include:

- `seeksAttn(?a)`: agent `?a` seeks attention,
- `greeted(?a)`: agent `?a` has been greeted,
- `ordered(?a)`: agent `?a` has ordered,
- `ackOrder(?a)`: agent `?a`'s order has been acknowledged,
- `served(?a)`: agent `?a` has been served,
- `otherAttnReq`: other agents are seeking attention,
- `badASR(?a)`: agent `?a` was not understood, and
- `transEnd(?a)`: the transaction with `?a` has ended.

Two functions are also defined:

- `inTrans = ?a`: the robot is interacting with `?a`, and
- `request(?a) = ?d`: agent `?a` has requested drink `?d`.

We use a typed version of the domain with two types: `agent` and `drink`. All predicate arguments accept constants of type `agent`, while `inTrans` maps to type `agent`, and `request` takes an argument of type `agent` and maps to type `drink`.

```
action greet(?a : agent)
   preconds: K(inTrans = nil) & ¬K(greeted(?a)) &
             K(seeksAttn(?a)) & ¬K(ordered(?a)) &
             ¬K(otherAttnReq) & ¬K(badASR(?a))
   effects:  add(Kf,greeted(?a)),
             add(Kf,inTrans = ?a)

action ask-drink(?a : agent)
   preconds: K(inTrans = ?a) & ¬K(ordered(?a))
             ¬K(otherAttnReq) & ¬K(badASR(?a)) &
   effects:  add(Kf,ordered(?a)),
             add(Kv,request(?a))

action ack-order(?a : agent)
   preconds: K(inTrans = ?a) & K(ordered(?a)) &
             ¬K(ackOrder(?a)) & ¬K(otherAttnReq) &
             ¬K(badASR(?a))
   effects:  add(Kf,ackOrder(?a))

action serve(?a : agent, ?d : drink)
   preconds: K(inTrans = ?a) & K(ordered(?a)) &
             Kv(request(?a)) & K(request(?a) = ?d) &
             K(ackOrder(?a)) & ¬K(otherAttnReq) &
             ¬K(badASR(?a))
   effects:  add(Kf,served(?a))

action bye(?a : agent)
   preconds: K(inTrans = ?a) & K(served(?a)) &
             ¬K(otherAttnReq) & ¬K(badASR(?a))
   effects:  add(Kf,transEnd(?a)),
             add(Kf,inTrans = nil)

action not-understand(?a : agent)
   preconds: K(inTrans = ?a) & K(badASR(?a))
   effects:  del(Kf,badASR(?a))
```

Figure 4: PKS actions in a single agent interaction

Actions in the domain are defined using the above fluents. In particular, our domain includes eight high-level actions:

- `greet(?a)`: greet an agent `?a`,
- `ask-drink(?a)`: ask agent `?a` for a drink order,
- `ack-order(?a)`: acknowledge agent `?a`'s drink order,
- `serve(?a,?d)`: serve drink `?d` to agent `?a`,
- `bye(?a)`: end an interaction with agent `?a`,
- `not-understand(?a)`: alert agent `?a` that its utterance was not understood, and
- `wait(?a)`: tell agent `?a` to wait, and
- `ack-wait(?a)`: thank agent `?a` for waiting.

Definitions for the first six actions (required for single agent interactions) are given in Figure 4. Actions are described at an abstract level and include a mix of physical, sensory, and speech acts. For instance, `serve` is a standard planning action with a deterministic effect (i.e., it adds definite knowledge to PKS's K_f database); however, when executed it causes the robot to hand over a drink to an agent and confirm the drink order through speech. Actions like `greet`, `ack-order`, and `bye` are modelled in a similar way, but only map to speech output at run time (e.g., "hello", "okay", and "good-bye"). The most interesting action is `ask-drink` which is modelled as a sensing action: the function term `request` is added to the planner's K_v database as an effect, indicating that this piece of information will become known at execution time. The `not-understand` action is used as a directive to the speech output system to produce an utterance that (hopefully) causes the agent to repeat its last response. The `wait` and `ack-wait` actions control interactions when multiple agents are seeking the attention of the bartender.

Finally, we also require a description of the domain's initial state and goal before we can build plans. The initial state, which includes a list of the objects (drinks) and agents (customers) in the bar, is not hard-coded in the domain description. Instead, this information is supplied to the planner by the state manager. Changes in the object or agent list are also sent to the planner, causing it to update its domain model. The `inTrans` function is initially set to `nil` to indicate that the robot isn't interacting with any agent. The planner's goal is simply to serve each agent seeking attention, i.e.,

```
forallK(?a : agent)
      K(seeksAttn(?a)) => K(transEnd(?a)).
```

This goal is viewed as a rolling target which is reassessed each time a state report is received from the state manager. We now consider some plans we can generate in this domain.

Ordering a drink: We first consider the case where there is a single agent `a1`. No specific drinks are defined and no other state information is supplied, except that the robot is not interacting with any agent (i.e., `inTrans = nil` $\in K_f$). The appearance of `a1` seeking attention is reported to PKS in an initial state report, which has the effect of adding a new constant named `a1` of type agent to the planner's domain description, and adding a new fact `seeksAttn(a1)` to the initial K_f database. Using this initial state and the above actions, PKS can build the following plan to achieve the goal:

`greet(a1)`,	[Greet agent `a1`]
`ask-drink(a1)`,	[Ask `a1` for drink order]
`ack-order(a1)`,	[Acknowledge `a1`'s drink order]
`serve(a1,request(a1))`,	[Give the drink to `a1`]
`bye(a1)`.	[End the transaction]

Initially, the planner can choose `greet(a1)` since `inTrans = nil` $\in K_f$ and `seeksAttn(a1)` $\in K_f$, and the other preconditions are trivially satisfied (i.e., none of `greeted(a1)`, `ordered(a1)`, `otherAttnReq`, or `badASR(a1)` are in K_f). After `greet(a1)`, the planner is in a state where `inTrans = a1` $\in K_f$ and `greeted(a1)` $\in K_f$. The `ask-drink(a1)` action can now be chosen, updating PKS's knowledge state so that `ordered(a1)` $\in K_f$ and `request(a1)` $\in K_v$. The next action considered by the planner is `ack-order(a1)`, in particular since `ackOrder(a1)` $\notin K_f$. As a result `ackOrder(a1)` is added to K_f. Consider the `serve(a1,request(a1))` action. Since `inTrans = a1` remains in K_f, the first precondition of the action is satisfied. Since `ordered(a1)` $\in K_f$, the second precondition, `K(ordered(a1))`, holds. Also, since `request(a1)` $\in K_v$, the third precondition `Kv(request(a1))` holds (i.e., the value of `request(a1)` is known). The fourth precondition, `K(request(a1)=request(a1))` is trivially satisfied since both sides of the equality are syntactically equal; this also has the effect of binding `request(a1)` to `serve`'s second

parameter. Thus, request(a1) acts as a run-time variable whose definite value (i.e., a1's drink order) will become known at run time. The fifth precondition is satisfied by the effects of ack-order(a1). The remaining two preconditions are also trivially satisfied. The action updates K_f so that served(a1) $\in K_f$, leaving K_v unchanged. Finally, bye(a1) is added to the plan resulting in inTrans = nil $\in K_f$ and transEnd(a1) $\in K_f$, satisfying the goal.

Ordering a drink with restricted drink choices: The above plan relies on PKS's ability to use function terms as run-time variables in parameterised plans. However, doing so requires additional reasoning, potentially slowing down plan generation in domains where many such properties must be considered. Furthermore, it does not restrict the possible mappings for request, except that it must be a drink.

Consider a second example, again with a single agent a1 seeking attention, where PKS is also told there are three possible drinks that can be ordered: juice, water, and beer. In this case, the drinks are represented as constants of type drink, i.e., juice, water, and beer. Information about the drink options is also put into PKS's K_x database as the formula:

```
(request(a1) = juice | request(a1) = water |
                request(a1) = beer),
```

which restricts the possible mappings for request(a1). PKS can now build a plan of the form:

```
greet(a1),                      [Greet agent a1]
ask-drink(a1),                  [Ask a1 for drink order]
ack-order(a1),                  [Acknowledge a1's order]
branch(request(a1))             [Form branching plan]
   K(request(a1) = juice):      [If order is juice]
      serve(a1,juice)           [Serve juice to a1]
   K(request(a1) = water):      [If order is water]
      serve(a1,water)           [Serve water to a1]
   K(request(a1) = beer):       [If order is beer]
      serve(a1,beer)            [Serve beer to a1]
bye(a1).                        [End the transaction]
```

In this case, a conditional plan is built with branches for each possible mapping of request(a1). E.g., in the first branch request(a1) = juice is assumed to be in the K_f database; in the second branch request(a1) = water is in K_f; and so on. Planning continues in each branch under each assumption. (We note that this type of branching was only possible because the planner had initial K_x knowledge that restricted request(a1), combined with K_v knowledge provided by the ask-drink action.) Along each branch, an appropriate serve action is added to deliver the appropriate drink. In more complex domains (currently under development), each branch may require different actions to serve a drink, such as putting the drink in a special glass or interacting further with the agent using additional information gathering actions (i.e., "would you like ice in your water?").

Ordering drinks with multiple agents: Our simple planning domain also enables more than one agent to be served if the state manager reports multiple customers are seeking attention. For instance, say that there are two agents, a1 and a2 (as in Figure 2). One possible plan that might be built is:

```
wait(a2),                       [Tell agent a2 to wait]
greet(a1),                      [Greet agent a1]
ask-drink(a1),                  [Ask a1 for drink order]
ack-order(a1),                  [Acknowledge a1's drink order]
serve(a1,request(a1)),          [Give the drink to a1]
bye(a1),                        [End a1's transaction]
ack-wait(a2),                   [Thank a2 for waiting]
ask-drink(a2),                  [Ask a2 for drink order]
ack-order(a1),                  [Acknowledge a2's drink order]
serve(a2,request(a2)),          [Give the drink to a2]
bye(a2).                        [End a2's transaction]
```

Thus, a1's drink order is taken and processed, followed by a2's order. The wait and ack-wait actions (which aren't needed in the single agent plan) are used to defer a transaction with a2 until a1's transaction has finished. (The otherAttnReq property, which is a derived property defined in terms of seeksAttn, ensures that other agents seeking attention are told to wait before an agent is served.)

One drawback of our domain encoding is that agents who are asked to wait are not necessarily served in the order they are deferred. From a task achievement point of view, such plans still succeed in their goal of serving drinks to all agents. However, from a social interaction point of view they potentially fail to be appropriate (depending on local pub culture), since some agents may be served before others that have been waiting for longer periods of time. This situation can arise in our domain since the appearance of a new agent is dynamically reported to the planner, possibly triggering a replanning operation: the newly built plan might preempt a waiting agent for a newly-arrived agent as the next customer for the bartender to serve. Since socially appropriate interaction is central to this work, we are modifying our domain to include ordering constraints on waiting agents.

Low-confidence utterances and overanswering: Once a plan is built, it is executed by the robot, one action at a time. Each action is divided into head, speech, and arm behaviours based on a simple set of rules, before it is executed in the real world. At run time, PKS's execution monitor assesses plan correctness by comparing subsequent state reports from the state manager against states predicted by the planner. In the case of disagreement, for instance due to unexpected outcomes like action failure, the planner is invoked to construct a new plan using the sensed state as its new initial state. This method is particularly useful for responding to unexpected responses by agents interacting with the bartender.

For example, if the planner receives a report that a1's response to ask-drink(a1) was not understood, for instance due to low-confidence automatic speech recognition, the state report sent to PKS will have no value for request(a1), and badASR(a1) will also appear. This will be detected by the monitor and PKS will be directed to build a new plan. One result is a modified version of the original plan that first informs a1 they were not understood before repeating the ask-drink action and continuing the old plan:

```
not-understand(a1),     [Alert a1 it was not understood]
ask-drink(a1),          [Ask a1 again for drink order]
...continue with remainder of old plan...
```

Thus, replanning produces a loop that repeats an action in

an attempt to obtain the information the planner requires in order to continue executing the previous plan.

Another useful consequence of this approach is that certain types of over-answering by the interacting agent can be handled by the execution monitor through replanning. For instance, a `greet(a1)` action by the bartender might cause the customer to respond with an utterance that includes a drink order. In this case, the state manager would include an appropriate `request(a1)` mapping in the state description, along with `ordered(a1)`. The monitor would detect that the preconditions for `ask-drink(a1)` aren't met and direct PKS to replan. A new plan could then omit `ask-drink` and proceed to acknowledge and serve the requested drink.

Extended Reasoning for Dialogue Planning

Although our simple bartending domain handles a number of realistic interactions, we are also extending it and the underlying planner to model more complex scenarios which arise in dialogue-based interactions with human agents.

First, we are exploring the idea of passing an n-best list of processed automatic speech recognition (ASR) hypotheses to the state manager for inclusion in the state representation, as a set of alternative interpretations for an agent's utterance. In practical terms, the n-best can be determined by the list of top entries that account for a significant probability mass in terms of the hypotheses' associated confidence measures, i.e., $\{\langle h_1, c_1\rangle, \langle h_2, c_2\rangle, \ldots \langle h_n, c_n\rangle\}$, such that $\sum_{i=1}^{n} c_i > \theta$, where θ is some threshold. Using this list, the state manager can then derive a set of interpretations, $\{\phi_1, \phi_2, \ldots, \phi_n\}$, where each ϕ_i is a conjunction of state fluents. (In our domain, each ϕ_i is usually a single fluent.)

At the planning level, such disjunctive state information can be represented in PKS's K_x database as an "exclusive or" formula of the form $(\phi_1|\phi_2|\ldots|\phi_n)$. Once such information is available in the planner's knowledge state, it can be used during plan construction. In practice, such knowledge often has the effect of requiring extra actions in a plan, to disambiguate between K_x alternatives. To aid in this process, we are adding new sensing actions (corresponding to questions the robot can ask), to help clarify uncertain beliefs.

Second, we are also extending PKS's ability to work with certain types of multiagent knowledge that arise in dialogue scenarios (Steedman and Petrick 2007). For instance, *common ground* information (i.e., the beliefs about the conversational state that arise during an interaction) is often used by mainstream dialogue systems but is not present in our domain. Currently, PKS cannot easily model other agents' beliefs; rather, such information must be encoded using the standard (single agent) tools available in PKS. We are adding new operators to PKS that enable it to directly represent and reason with such knowledge during planning. This is particularly important in dialogue contexts, since more effective dialogue moves can often be made by taking into consideration what other agents believe about an interaction.

Currently, in the ideal case, our simple planning domain guides the robot's interaction through a relatively fixed sequence of actions derived from studies of human-human interaction in real bars (Huth 2011). The next version of the system will extend this necessary baseline by adding more flexibility to the drink ordering process (see below), giving the planner and execution monitor greater scope. The addition of multiagent knowledge in PKS will also enable the planner to reason about agent intentions and beliefs in a less rigid way, letting us plan with more domain-independent speech acts. However, even the present, simple scenario has proven useful for testing off-the-shelf planning tools in place of more specialised approaches to interaction management.

Discussion and Related Work

We have tested our basic bartending domain with human users in a drink ordering scenario. In particular, we carried out a user evaluation in which 31 participants interacted with the robot bartender in three separate social interactions involving multiple agents. Overall, most customers (95%) successfully ordered a drink from the bartender, and the robot dealt appropriately with multiple simultaneous customers and with unexpected situations of the form described above (i.e., over-answering and input-processing failure). More details of this evaluation are available in (Foster et al. 2012).

The general focus of this work is *social robotics*, which extends traditional robotics by situating robots in social and cultural contexts. In contrast to recent work in this field, we address a different style of interaction which is distinctive in two ways. First, existing projects generally consider social interaction as the primary goal, while our robot supports social communication in the context of cooperative, task-based interaction. Second, while most social robotics systems deal primarily with one-on-one interactive situations, such as building long-term companion relationships (Breazeal 2005; Dautenhahn 2007), our bartender must deal with dynamic, multiagent scenarios: people will constantly enter and leave the scene, so the robot must continually choose appropriate social behaviour while interacting with new partners.

The use of planning is central to this work, an idea with a long tradition in natural language generation and dialogue. Early approaches to generation as planning (Perrault and Allen 1980; Appelt 1985; Young and Moore 1994) focused primarily on high-level structures, such as speech acts and discourse relations, but suffered due to the inefficiency of the planners available at the time. As a result, recent mainstream research has tended to segregate task planning from discourse and dialogue planning, capturing the latter with specialised approaches such as finite state machines, information states, speech-act theories, or dialogue games (Traum and Allen 1992; Matheson, Poesio, and Traum 2000; Asher and Lascarides 2003; Maudet 2004).

Recently, there has been renewed interest in applying planning methods to natural language problems such as sentence planning (Koller and Stone 2007), instruction giving (Koller and Petrick 2011), and accommodation (Benotti 2008). The idea of treating interaction management as planning with incomplete information and sensing has also been revisited (Stone 2000), a view that is implicit in early BDI-based approaches, e.g., (Litman and Allen 1987; Cohen and Levesque 1990; Grosz and Sidner 1990). Initial work using PKS also explored this connection (Steedman and Petrick 2007), but fell short of implementing an efficient tool. A related approach (Brenner and Kruijff-Korbayová 2008) man-

ages dialogues by interleaving planning and execution, but fails to solve the problem of deciding when best to commit to plan execution versus plan construction. Thus, many recent planning approaches are promising, but not yet fully mature, and fall outside of mainstream interactive systems research.

Conclusions and Future Work

We presented an approach to task-based social interaction in a robot bartender domain, using knowledge-level planning techniques. Actions are selected by the PKS planner, using states derived from low-level visual and speech inputs. Plans are executed on a robot platform to control an animatronic head, speech synthesiser, and a pair of manipulator arms.

We are currently extending the bartending domain to support more complex interactions, including agents that can ask questions about drinks, a bartender that can query agents for more information, and groups of agents that can order multiple drinks. Managing such scenarios will require additional support from the state manager. To address this, we will make use of supervised learning techniques trained on data gathered from humans interacting with both real and artificial bartenders, e.g., using methods similar to those of (Bohus and Horvitz 2009). Based on the results of our initial evaluation, we believe that general-purpose planning continues to be a promising tool for developing task-based interactive systems, as an alternative to specialised approaches, such as those used in many dialogue systems.

Acknowledgements

The authors thank their colleagues from the JAMES Project who helped implement the bartender system: Andre Gaschler and Manuel Giuliani from fortiss Gmbh, Maria Pateraki from FORTH-Hellas, and Amy Isard and Richard Tobin from the University of Edinburgh. This research has received funding from the European Union's 7th Framework Programme (FP7/2007–2013) under grant No. 270435.

References

Appelt, D. 1985. *Planning English Sentences*. Cambridge University Press.

Asher, N., and Lascarides, A. 2003. *Logics of Conversation*. Cambridge University Press.

Baltzakis, H.; Pateraki, M.; and Trahanias, P. 2012. Visual tracking of hands, faces and facial features of multiple persons. *Machine Vision and Applications* 1–17.

Benotti, L. 2008. Accommodation through tacit sensing. In *Proceedings of SemDial-2008 (LONDIAL)*, 75–82.

Bohus, D., and Horvitz, E. 2009. Dialog in the open world: platform and applications. In *Proceedings of ICMI-2009*, 31–38.

Breazeal, C. 2005. Socially intelligent robots. *interactions* 12(2):19–22.

Brenner, M., and Kruijff-Korbayová, I. 2008. A continual multiagent planning approach to situated dialogue. In *Proceedings of SemDial-2008 (LONDIAL)*, 67–74.

Cohen, P., and Levesque, H. 1990. Rational interaction as the basis for communication. In *Intentions in Communication*. MIT Press. 221–255.

Dautenhahn, K. 2007. Socially intelligent robots: dimensions of human-robot interaction. *Philosophical Transactions of the Royal Society B: Biological Sciences* 362(1480):679–704.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Foster, M. E.; Gaschler, A.; Giuliani, M.; Isard, A.; Pateraki, M.; and Petrick, R. P. A. 2012. Two people walk into a bar: Dynamic multi-party social interaction with a robot agent. In *Proceedings of ICMI-2012*.

Grosz, B., and Sidner, C. 1990. Plans for discourse. In *Intentions in Communication*. MIT Press. 417–444.

Huth, K. 2011. Wie man ein Bier bestellt. MA thesis, Fakultät für Linguistik und Literaturwissenschaft, Universität Bielefeld.

Isard, A., and Matheson, C. 2012. Rhetorical structure for natural language generation in dialogue. In *Proceedings of SemDial-2012 (SeineDial)*, 161–162.

Koller, A., and Petrick, R. P. A. 2011. Experiences with planning for natural language generation. *Computational Intelligence* 27(1):23–40.

Koller, A., and Stone, M. 2007. Sentence generation as planning. In *Proceedings of ACL-2007*, 336–343.

Litman, D., and Allen, J. 1987. A plan recognition model for subdialogues in conversation. *Cognitive Science* 11:163–200.

Mann, W. C., and Thompson, S. A. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text* 8(3):243–281.

Matheson, C.; Poesio, M.; and Traum, D. 2000. Modeling grounding and discourse obligations using update rules. In *Proceedings of NAACL-2000*.

Maudet, N. 2004. Negotiating language games. *Autonomous Agents and Multi-Agent Systems* 7:229–233.

Perrault, C. R., and Allen, J. F. 1980. A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics* 6(3–4):167–182.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS-2002*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of ICAPS-2004*, 2–11.

Steedman, M., and Petrick, R. P. A. 2007. Planning dialog actions. In *Proceedings of SIGdial-2007*, 265–272.

Steedman, M. 2000. *The Syntactic Process*. MIT Press.

Stone, M. 2000. Towards a computational account of knowledge, action and inference in instructions. *Journal of Language and Computation* 1:231–246.

Traum, D., and Allen, J. 1992. A speech acts approach to grounding in conversation. In *Proceedings of ICSLP-1992*, 137–140.

Vinciarelli, A.; Pantic, M.; and Bourlard, H. 2009. Social signal processing: Survey of an emerging domain. *Image and Vision Computing* 27(12):1743–1759.

White, M. 2006. Efficient realization of coordinate structures in Combinatory Categorial Grammar. *Research on Language and Computation* 4(1):39–75.

Young, R. M., and Moore, J. D. 1994. DPOCL: a principled approach to discourse planning. In *Proceedings of the International Workshop on Natural Language Generation*, 13–20.